

THE SOFTWARE ENGINEERING GLOBAL MODEL

CLAUDINE TOFFOLON (*) (**)

SALEM DAKHLI (**)

(*) LIL, LITTORAL UNIVERSITY, Calais, France

(**) CERIA, PARIS-DAUPHINE UNIVERSITY, France

E-mail: Claudine.Toffolon@dauphine.fr

E-mail: sdakhli@computer.org

Keywords: agency theory, software crisis, software dimensions, transaction costs theory, project spaces.

Abstract: Despite the economic and functional importance of software systems in modern organizations, software development and maintenance are still regarded as high-risk activities. Indeed, a great number of software projects fail to deliver software systems meeting stakeholders requirements within schedule and budget. So, the software industry is in crisis called the “software crisis”. The great number of software project failures and the high maintenance costs are the economic ramifications of the “software crisis”. The social ramifications of this crisis are related to the user resistance to software systems use within organizations. This paper provides a framework, called the “software engineering global model” composed of two parts – static and dynamic - which provides basic foundations to build instruments eliminating the causes the “software crisis” and reducing its impacts.

1. INTRODUCTION

Information technology plays a critical role in modern organizations. Indeed, to face the increasing globalization of markets and the constraints of changing economic and technological environments, organizations use information technology as instrument of competitiveness. In particular, contemporary information systems provide the main tools to support business and decision making processes within organizations whose existence is strongly dependent on software systems. Many authors have illustrated the organization dependence on information technology. For example, Boehm et al. (1988) projected that computer software costs will grow to more than \$800 billions worldwide by 2000, and

estimated for 1990 that these costs will represent 13% of the U.S. GNP. According to Willcocks (1994), the whole IT industry will account for some 10% of the world economic activity by the year 2000. This figure is about two times the 1990 figure. This author noted a rising trend in IT expenditure over the last two decades. Gurbaxani et al. (1991) used the economic agency and transaction cost theories to demonstrate that information technology shapes organizations by determining their optimal size and helping them to optimize their decision making process. On the basis of a detailed analysis of organizations cost structure, these authors show that information technology has both centralized the decision making process by lowering costs of information dissemination and decentralized this process by lowering monitoring costs. In addition, by reducing internal coordination

costs, information technology increases organization's vertical size, which consists in the number of chains of value adding covered. By increasing network externalities and economies of scale with manufacturing technology, information technology leads to larger firms horizontally i.e. in terms of number of markets covered and market share therein. Gurbaxani et al. (1991) noticed however that information technology could also lead to vertically smaller firms due to lower external costs.

Nevertheless, the value got out of organizations vast investments in information technology is less than expected since software industry is in a crisis called "the software crisis". This crisis, which has been with us for more than thirty years, is one of causes of the "productivity paradox" pointed out by Solow (1987) and analyzed by many authors (Brynjolfsson 1993, Brynjolfsson 1998, Willcocks 1999). In this paper, we propose a framework, called "the software engineering global model" which provides instruments to deal with the "software crisis" and reduce its negative impacts. Our paper is organized as follows. In section 2, we present the main characteristics of "the software crisis". Section 3 describes the foundations of the "software engineering global model" presented in section 4. Section 5 provides a list of applications of "the software engineering global model" to the software engineering discipline. In section 6, we conclude this paper by describing lessons learned and future research directions.

2. THE "SOFTWARE CRISIS"

The term "software crisis" has been frequently used since the 60's to allude to a set of problems encountered in software engineering. To underline the chronic nature of this phenomenon, Pressman (1997) suggests the use of the term "chronic affliction" rather "crisis". The "software crisis" is characterized by two categories of problems associated either with the software artifacts development, maintenance and management processes, or with software products use. These problems are

related to inaccuracy of schedule and cost estimates, low productivity of software developers and poor quality of software products. The "software crisis" has two ramifications: social and economic. Social ramifications of the "software crisis" are related to user resistance to implementation and use of software systems within organizations. User resistance is mainly due to organizational and social changes induced by new software systems and related to the role of information technology within organizations. Economic ramifications of the "software crisis" result from the great number of software failures and the high maintenance costs of software systems. Indeed, a great number of software projects fail to deliver software systems meeting stakeholders requirements within schedule and budget. Viewed another way, maintenance of software systems amounts to about 60 to 85% of total software costs in industry (Wiederhold 1995). According to the Standish Group research (Standish 1995), yearly expenditures on software systems development in the United States were more than \$250 billion for approximately 175,000 projects. The average cost of a software project is \$2,322,000 for a large company, \$1,331,000 for a medium company and \$434,000 for a small company. This study shows that:

- about 31% of projects are canceled before completion,
- 53% of projects cost more than 180% of their original estimated cost,
- in 1995, American organizations spent about \$81 billion for canceled projects and additional \$59 billion to complete software projects.
- the average percentage of software projects completed on time and on budget is only 16% for all organizations and 9% for large organizations.

The known estimations of the costs of software project failures and overruns are just the tip of the iceberg. Indeed, software project cancellation, cost overruns, or late delivery result in losing important but not measurable

opportunity costs. To realize the extent of this problem, let us note that the failure to develop a reliable software system to handle baggage at the new Denver International airport reached \$1.1 million per day. Solutions proposed to deal with the “software crisis” are presented synthetically in the next section.

3. SOLUTIONS OF THE “SOFTWARE CRISIS”

Two categories of solutions are proposed to date in order to deal with the “software crisis”. On the one hand, there are approaches for reducing the amount of work to be done by software developers. These approaches are based upon end-user computing and standard application packages. On the other hand, there are approaches for improving developers’ productivity by raising the software development process efficiency and effectiveness. Efficiency improvement results from use of CASE tools and software engineering environments while effectiveness improvement is related to the quality of software systems requirements. All these solutions have partly failed, firstly because of the increasing complexity of software systems and secondly, because they don’t take into account all the aspects of software engineering (Toffolon 1996). In particular, these solutions don’t focus sufficiently on the real causes of the “software crisis”. That doesn’t mean that paradigms, methods, tools, and technologies used to date are bad. We think that they provide efficient instruments to deal with some aspects of the “software crisis”, if the real problems are identified and well known. That the software engineering global model proposed in this paper is a framework, which provides instruments to build appropriate solutions to the “software crisis”.

4. THE FOUNDATIONS OF THE SOFTWARE ENGINEERING GLOBAL MODEL

Well-known definitions of software engineering often stress the technical aspects of this discipline while neglecting totally or partly its organizational, social and economic aspects. For example, software engineering definition provided by Nauer et al. (1968) does not take into account either social and organizational aspects of software engineering or economic aspects of the software products. Use of this traditional view of software engineering as the basis of known solutions to the “software crisis” results in many weaknesses inherent in these solutions. To take into account of software engineering, we define software engineering as a set of organization’s actors, which use human, technical and informational resources to built a high quality computer solution to an organizational problem, on-time and on-budget. This definition stresses many ideas. Firstly, software engineering involves many actors within organization actors and is not only the developers business. Secondly, the goal of software engineering is the computerization of well-defined organizational solutions in order to provide a software product aimed to support organization’s business and operational processes. Finally, organizational, social and economic aspects are inherent at the same time in the software development process and the product issued from this process. Consequently, the definition of solutions to the “software crisis” must be preceded by the identification of all the aspects of software engineering, actors involved, and resources used and produced by this discipline. To reach this goal, we consider that software engineering is characterized by four facets: business-oriented (also called structure-oriented), actors-oriented, behavior-oriented and information-oriented. The four facets of software engineering correspond to the four major abstraction levels used to solve complex problems: the conceptual level (What?), the organizational level (Who?), the logical level

(With?) and the physical level (How?). Firstly, by stressing information related to software engineering processes and software artifacts issued from this process, the “information-oriented” facet corresponds to the conceptual level. Secondly, the “actors-oriented” facet corresponds to the organizational level since it provides information related to software engineering actors and the roles they play within organizations. Thirdly, businesses defined by the “business-oriented” facet permit describing tools associated with the logical level. Finally, the “behavior-oriented” facet corresponds to the physical level since actors’ behavior defined by this facet indicates how to built computer solution to organizational problems.

To identify and describe these four facets, we use the by transaction cost theory (Coase 1937, Williamson 1981), the agency theory (Alchian et al. 1972) and the “software dimensions theory” (Toffolon 1999). Although actors have been identified as an essential component of modern organizations by many organization’s models like for example the Leavitt’s model (Leavitt 1963), the economic agency theory provides more accurate information about actors interactions and behavior within organizations. Such information permits identifying the “actors-oriented” and the “behavior-oriented” facets of software engineering. The “business-oriented” facet is determined using the transaction cost theory which identifies in particular actors action fields within organizations. The “information-oriented” facet definition is based on the software dimensions theory.

4.1 The agency theory

Economists have realized that the neo-classical assumption that a firm behaves as a team, which maximize its profits has limited scope and is inadequate in analyzing managerial behavior within modern organizations. They propose an economic theory, called agency theory, to explain the situations where managerial behavior is inconsistent with profit maximization.

Agency theory (Alchian et al. 1972) analyzes an organization as a nexus of contracts among self-interested individuals. Each agency contract links a principal (entrepreneur) and agents (employees) in order to perform some service. Agency theory is based on the following assumption: each agent maximizes its proper utility and pays no regard to the welfare of the principal or non-pecuniary virtues. The discrepancies between the objectives of the principal and those agents result in agency costs which are the sum of monitoring costs, bonding costs (documentation and reporting costs from agent to principal) and residual loss. So, agency theory permits understanding the structure of the organization’s internal coordination costs composed of agency costs and decision-information costs. Agency costs concern the acquisition of information on agents’ behavior while decision-information costs reflect effort and time needed to search and process information related to decision-making. The agency theory provides useful insights into organizations and permits explaining how an organization can be. It focuses on problems related to informational considerations and provides instruments to overcome them so those firms are viable forms of economic organization. Nevertheless, the agency theory doesn’t explain why an organization can be under certain conditions a more efficient form of institution than a market. The transaction cost theory provides tools to answer this question.

Applying agency theory in analyzing information technology role in modern organizations demonstrates that software engineering is governed by a set of contracts among actors concerned with the software system to be developed or maintained. At a given time, each project actor plays the role of consumer (principal) or producer (agent) under the contracts, which link him to the other project actors. So, a software development or maintenance project is a nexus of contracts among different actors with conflicting interests and points of view. The discrepancies between the project actors’ objectives are

partly the source of software engineering inconsistencies and related agency costs. A preliminary condition to cope with these inconsistencies is to identify the different actors concerned with the software project as well as their objectives (e.g. what they expect from the project). Consequently, according to the economic agency theory, the “behavior-oriented” facet of software engineering is characterized by enforcement of contracts between self-interested actors involved in this discipline.

4.2 The transaction costs theory

The transaction cost theory (Coase 1937, Williamson 1981) analyzes markets problems and views firms as solutions to these problems. It considers that market operations are not costless as is assumed in classical economic theory. Market operations result in transaction cost which are the costs of administering an exchange relationship, a transfer of some good or service between technologically separable activities. Transaction cost include the costs of negotiating, drafting, and monitoring contracts; the costs of settling disputes and enforcing settlements, and the opportunity costs related to inefficient administration of contracts before reaching a necessary new agreement. According to this theory, assessment of transaction costs demonstrates that the firm is a substitute for the market mechanism, created to reduce transaction costs. Transaction costs theory helps understanding how markets and hierarchies are chosen. According to Malone et al. (1987) and Picot et al. (1987), markets and hierarchies are two basic mechanisms for coordinating the flow of materials and services through adjacent steps in the value chain. (Williamson 1981) distinguishes two categories of transactions: market transactions and hierarchy transactions. The first category support coordination between multiple buyers and sellers while the second category permits coordination within the firm as well as industry value chain.

Dakhli et al. (1997) notices that confusion between many businesses is one of the major weaknesses of well-established software

development methodologies. At least two factors make clear this confusion. Firstly, software development methodologies in use are built upon the same principles than the manufacturing processes characterized by well-known problems and solutions. Such methodologies are appropriate to develop software systems to computerize well-structured and repetitive tasks like accounting. They are not efficient in developing software systems related to ill-structured problems like Decision Support Systems or Knowledge Management Systems. Secondly, because of the role played by Information Technology in modern organizations, software systems needed are accumulation of many types of knowledge: economic, organizational, technical, social...Therefore many businesses are involved in software products development, maintenance and use. Applying the transactions costs theory at the actors level permits concluding that actors contribution to software engineering activities must be compliant with their know-how and businesses in which they are the most efficient. To identify businesses involved in software engineering, we notice that the development of a software system is associated with two problem-solving processes. The organizational problem solving process consists in building organizational solution to a problem related to organization’s operational or business processes. This problem solving process is bottom-up and involves two types of actors: the end-user who defines the problem and the customer/organizer who builds the organizational solution. The software problem solving process consists in developing a computer solution to an organizational problem. In other terms, the software problem solving process permits total or partial computerization of an organizational solution. This process is top-down and involves two types of actors: the software architect who builds the computer solution and the software developer who implements it. Consequently, the “business-oriented” and “actors-oriented” facets of software engineering are characterized respectively by four major businesses and four types of actors involved in software

engineering activities. According, to the transaction cost theory, those businesses correspond to four mini-organizations called project spaces described below.

4.3 The software dimensions theory

(Toffolon 1999) identifies ten major aspects, or dimensions, of software. These dimensions have been determined on the basis of a deep analysis of the effects of the software crisis and organizations’ structure, that means interrelations between all organizational components (structural, tasks, individual, technical), environment and information technology.

Those ten dimensions concern altogether the software process and the artifacts produced by this process. The software process’ dimensions are cost dimension, delay dimension, technical dimension, communication dimension, and organizational dimension. The software product’s dimensions are functional dimension, human dimension, economic dimension, organizational dimension, and temporal dimension. Such dimensions demonstrate that the same software may reflect many different realities, which depend on the organizational, social and economic contexts of its use and exploitation.

The software dimensions theory provides instruments to take into account all aspects of software engineering and disseminate information needed to master the development, maintenance and use of software products. In that way, the ten dimensions permit characterizing the “information-oriented” facet of software engineering.

The software engineering four facets identified in this section permit us to model software engineering as activities carried out by self-interested organizational actors who use structures defined by their businesses to develop software artifacts in order to enforce contracts linking them. Figure 1 illustrates this model.

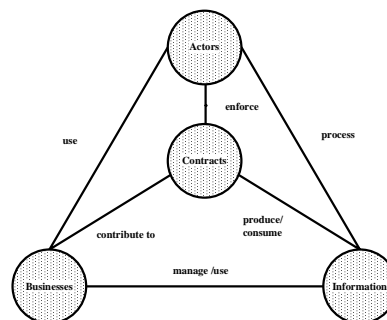


Figure 1: Software engineering representation

The transaction cost theory, the economic agency theory and the software dimensions theory play complementary roles in modeling software engineering. Indeed, the transaction cost theory permits identification of the actors and businesses involved in software engineering. The agency theory underlines the conflicting interests and points of view of the organization’s actors concerned with the software system, and provides an instrument to model software engineering in terms of contracts between consumers (principals) and producers (agents). The software dimensions theory permits identifying all the software engineering aspects and facilitates the expression of the organization’s priorities and constraints according to the Simon’s Bounded Rationality Principle (Simon 1983).

5. THE SOFTWARE ENGINEERING GLOBAL MODEL

The definition of the software engineering global model proposed in this paper rests on the description of links between entities characterizing the four facets of software engineering. This model is composed of two parts: static and dynamic. The static part – also called software engineering environment – refers to the four project spaces pointed out in a previous section while the dynamic part corresponds to the software engineering process.

5.1 The four project spaces

As noticed above, the four project spaces are assimilated to mini-organizations associated with actors and businesses involved in software engineering. Those spaces are:

- ❶ **The problem space** where are defined the customers and users problems and their organizational solutions. This space represents the stakeholder/customer's business.
- ❷ **The solution or architectural space** where are defined the computer solutions of the customer/user's problems. This space represents the architect's business.
- ❸ **The construction space** where these solutions are implemented. This space represents the developer's business.
- ❹ **The operation space** where are evaluated the software's usability from the user's perspective as well as its contribution to the organization's competitiveness. This space represents the end user's business.

Besides, each software project is represented in the four spaces by a static part, a dynamic part, and actors. In each space, project's dynamic part relates to the software engineering process, project's static part is composed of software artifacts resulting from this process, while project actors are human resources concerned with this project. Each actor may be either producer (agent) or consumer (principal) of software artifacts.

A role played by a project's actor in one of the four spaces is either principal or secondary. In each space, it is possible that a project has many actors assuming secondary roles, but there can be only one project actor involved in a principal role. Moreover, an actor can play a secondary role in many spaces, but a principal role only in one (every actor plays the principal role in some space). The project actors and the four spaces interrelations are illustrated by a tetrahedron (UCAD) where each face represents one space and each vertex represents a type of user (Figure 2). Each line of the tetrahedron (UCAD) shows the interaction between two project actors belonging to the

same space and each space is designated by a three letters acronym where the second letter corresponds to the project's actor playing the principal role in this space. According to this figure, the correspondences between spaces and actors are the following:

- In the problem space (UCA), the customer (C) plays the principal role while the user (U) and the architect (A) play secondary roles.
- In the solution (architectural) space (CAD), the architect (A) plays the principal role while the customer (C) and the developer (D) play secondary roles.
- In the construction space (ADU), the principal role is played by the developer (D) and the secondary roles are played by the architect (A) and the end user (U).
- In the operation space (DUC), the principal role is played by the user (U) and the secondary roles are played by the developer (D) and the customer (C).

The software dimensions are linked to the four project spaces since each project space is associated with a subset of the ten software dimensions. So,

- **the problem space** corresponds to the organizational, economic, temporal and functional software product dimensions,
- **the solution space** corresponds to the cost, delay, technical software process dimensions and to the functional, organizational software product dimensions,
- **the construction space** corresponds to the cost, delay, organizational and communication software process dimensions and to the functional and human software product dimensions,
- **the operation space** corresponds to the organizational, human and functional software product dimensions.

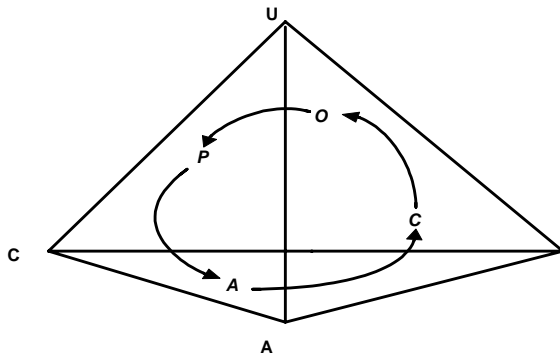


Figure 2: The four project spaces and associated actors

Figure 3 illustrates the correspondence between the four project spaces and the software dimensions. The four software project spaces provide tools to analyze deeply and describe the various contracts between the software project actors, as well as the “software crisis” in terms of deviations and inconsistencies inherent in software engineering.

	Spaces			
	Problem space	Solution space	Construction space	Operation space
Software product dimensions				
• Economic	✓			
• Temporal	✓			
• Organizational	✓	✓		✓
• Human			✓	✓
• Functional	✓	✓	✓	✓
Software development process dimensions				
• Cost		✓	✓	
• Delay		✓	✓	
• Organizational			✓	
• Communication			✓	
• Technical		✓		

Figure 3: The relationships between the four project spaces and the software dimensions

5.2 The dynamic part of the software engineering global model

In this section, we present the dynamic part of the software engineering global model from two perspectives.

The first perspective, also called meta-level, is based upon the “PACO” (Problem-Architecture-Construction-Operation) loop. It describes the correspondence between the four spaces and the tetrahedron’s (UCAD) sides. Such a correspondence originates from the

iterative progress of the software development process. Execution of the (PACO) loop proceeds as follows. The definition of a computer solution of an organizational problem permits the transition from the problem space to the solution space. The implementation of this solution expresses the transition from the solution space to the construction space. The installation of the software artifacts built in the construction space results in the transition from this space to the operation space. The description of problems and needs generated by the use of the software installed permits the

transition from the operation space to the problem space. The meta-level of the dynamic part of the software engineering global model rests on the roles played by the project's actors as human interfaces between project spaces. A project's actor who plays a principal role in one space and a secondary role in another space carries out the human interface between these two spaces. So:

- the human interface between the problem and the solution spaces is carried out by the customer (C) and the architect (A),
- the human interface between the solution and the construction spaces is carried out by the architect (A) and the developer (D),
- the human interface between the construction and the operation spaces is carried out by the developer (D) and the end user (U),
- the end user (U) and the customer (C) carry out the human interface between the operation and the problem spaces.

The second perspective, also called operational level, provides information related to the implementation of the software engineering process. To be complete and efficient, the description of the software engineering process must reflect the behavior of organizational actors while processing information related to enforcement of contracts linking them. Consequently, contribution of organizational actors to the software engineering process is associated with two types of activities: production and coordination. Production activities are carried out within each project space by the actor who plays the principal role in this space. Coordination activities are either intra-space activities or inter-spaces activities. In addition, to take into account changes induced by volatility and evolution of stakeholders needs, by technical and economic uncertainty, and by internal and external organization's constraints, the software engineering process associated with the dynamic part must be iterative at two levels. The first level, called intra-space iteration

level, relates to the progress of the processes supporting each project space. The second level, called inter-spaces iteration level, expresses that evaluation and coordination activities are iterative. Consequently, the proposed software development process is composed of five interdependent spiral models (Boehm 1988). (Figure 5) illustrates the five spirals.

1. A global spiral model, called coordination spiral, which permits inter-spaces "navigation", provides evaluation and coordination tools, and describes rules allowing intra-space and inter-spaces communication and roles distribution. This process is in compliance with the loop « Problem-Architecture-Construction-Operation » (**PACO**) (Figure 4). The coordination spiral takes place in the space generated by the ten software dimensions.
2. A spiral model called organizational spiral, which implements the organizational engineering process supporting the project's problem space. The organizational spiral is represented in the space generated by the software dimensions associated to the problem space.
3. A spiral model called architectural spiral, which implements the architecture definition process supporting the project's solution space. The architectural spiral is represented in the space generated by the software dimensions associated to the solution space.
4. A spiral model called software spiral, which implements the software implementation process supporting the project's construction space. The construction spiral is represented in the space generated by the software dimensions associated to the construction space.
5. A spiral model called operational spiral, which implements the software

systems use process supporting the project's operation space. The operational spiral is represented in the space generated by the software dimensions associated to the project's operation space.

Detailed description of the five spirals cited above is beyond the scope of this paper. The remainder of this section is dedicated to two important topics related to the dynamic part of the software engineering global model: Firstly, we provide a general description of the coordination activity related dynamic part of the software engineering global model. Then, we describe synthetically the resources used by the five spirals implementing the proposed software development process.

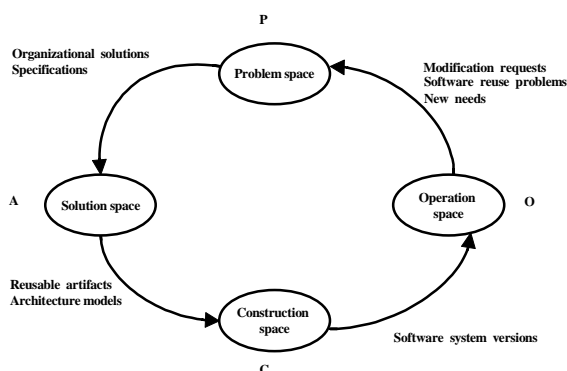


Figure 4: The (PACO) loop

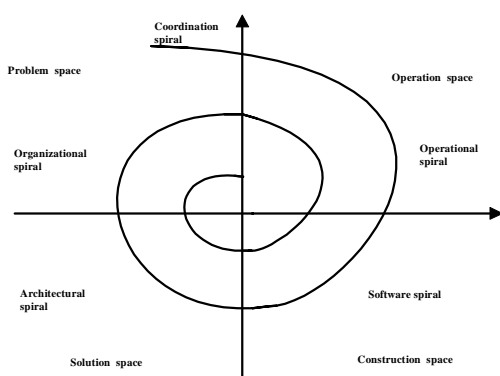


Figure 5: Software development process spirals

5.2.1 The coordination activity

The coordination activity, related to the dynamic part of the software engineering global model, takes place either within the four spaces or between those spaces. The coordination activity between two project's spaces is called horizontal coordination. The coordination activity within a project's space is called vertical coordination. The coordination spiral, which supports the horizontal coordination, is composed of two sub-processes: a communication sub-process and a meta-process. The communication sub-process describes the coordination tasks, the actors' communication rules, and the role distribution among project actors. The meta-process, based on a consumer/producer schema, describes negotiation rules and evaluation tasks; and defines, for software artifacts, the inter-spaces quality and acceptance criteria.

The coordination spiral uses two categories of resources contained in a repository common to all project spaces:

1. The first resources type relates to the evaluation methods, approaches, technologies, and tools in use within the organization. Cost/benefit methods, quality control techniques, tool selection methods are examples of such resources.
2. The second resources type relates to the organizational rules and models governing the relationships among project actors. They include in particular evaluation and acceptance criteria and thresholds used during meta-process implementation.

The vertical coordination aims at facilitating the progress of each spiral's cycle, the transition between two cycles, and the activities/teams coordination. We notice that, for each process supporting a project's space, a spiral's cycle consists in performing a service related to the organization's business or in elaborating a version of an organizational, an architectural, or a software solution.

The vertical coordination is supported by the four spaces' meta-processes. A project's space meta-process refers to evaluation, coordination and decision tasks within a project's space. A project's space meta-process is an integral part of the spiral model supporting this space. Each space's meta-process uses two types of resources: common resources shared with the coordination spiral and specific resources reflecting the space's characteristics and constraints. Common resources permit iterative improvement of software artifacts produced within each space, and provide tools to guarantee the acceptability of those artifacts needed by the other spaces. Specific resources permit each space's principal actor to take into account the space's organizational, technical and human contexts. Common evaluation and organizing resources provide tools to deal with horizontal communications problems while specific resources contribute to reduce vertical communication problems. So, each space's meta-process is a specialization of the coordination spiral. The number of iterations accomplished during the solution development is strongly correlated to the risk analysis and reduction tasks.

5.2.2 The dynamic part resources

Each spiral's cycle results in a set of software artifacts and intermediary products, which contribute to the implementation of a new version of a deliverable software product. They are constructed using either the specific resources of one project space or resources common to several spaces. For example, software lifecycle models, development methods and tools are resources shared by the architecture definition support process and the software implementation process. Resources used by the five processes are stored in a repository, called the corporate repository, composed of five layers, each one corresponding to an abstraction level:

1. the first level (level I) corresponds to the four project spaces specific resources,

2. the second level (level II) corresponds to the resources shared by two processes,
3. the third level (level III) corresponds to resources shared by three processes,
4. the fourth level (level IV) corresponds to resources shared by four processes,
5. the fifth level (level V) corresponds to resources shared by all processes.

Each abstraction level has two entries: a space entry that provide information on spaces using a given resource, and a resource entry that describes the type of this resource (method, tool, and model...). Detailed description of the corporate repository layers, as well as the inter-layers and intra-layers relationships, is provided in (Dakhli 1998).

6. APPLICATIONS OF THE SOFTWARE ENGINEERING GLOBAL MODEL

Many frameworks built upon the software engineering global model permits us to validate it in an operational environment. In the following we list three major applications of this model.

Firstly, we use this framework to define an organizational model of software artifacts maintenance and reuse within a French insurance company. This company which bought a set of business and technical reusable artifacts encountered many problems while defining rules to reuse, customize and manage those artifacts. The organizational model we suggest to cope with these problems is based on two processes. The first process is development for reuse process carried out by the software architect. The second process is the development with reuse process carried out by the software developer and the coordination process, which defines the inter-spaces and intra-space coordination rules between architects and developers. This framework was recently updated by defining a set of basic rules for configuration management of reusable software artifacts.

Secondly, we propose a framework to analyze the “software crisis” in terms of deviations and inconsistencies inherent in software engineering. This framework suggests solutions to reduce the “software crisis” impacts through the use of methods and tools compliant with organization’s maturity, know-how, constraints, and priorities. The proposed solutions are implemented in a medium-sized French company, which has many problems related poor quality of software products and high development and maintenance costs. Evaluation of this framework is in progress. Finally, the software engineering global model was used to elaborate an economic model of software reuse based on two layers: the first layer reflects the conflicting interests and points of view of actors involved in creation, customization and reuse of software artifacts. The second layer permits taking into account the corporate strategy, priorities and constraints. A prototype implementing this model has been developed.

7. CONCLUSION

The software engineering global model proposed is neither a panacea solution of the “software crisis” nor a silver bullet, but a framework which takes into account the major aspects and entities often omitted in the description of software engineering activities. This model provides basic foundations of a global architecture of software engineering environments and software artifacts development, maintenance and use processes. Validating the software engineering global model in real world operational environments demonstrates that it is open at two levels. On the one hand, since its dynamic part is based on Boehm’s spiral model, it facilitates knowledge capitalization by integrating all existing approaches, methods, techniques, and organizations’ know-how. On the other hand, it is appropriate to the main new software engineering paradigms e.g. cooperative work, intelligent agents systems, workflow technology...For example, a multi-agent

systems development methodology based on this model is under construction.

Finally, we notice that we have used the basic idea of this work as a starting point to build a global model of electronic commerce (Toffolon et al. 2002).

Nevertheless, the framework proposed in this paper can be improved in several ways. In particular, the detailed description of the “PACO” loop and the five spirals supporting the dynamic part of the software engineering global model seems to be a possible improvement of this work.

8. REFERENCES

- Alchian A.A, Demsetz H., 1972, “*Production, Information Costs and Economic Organization*”, *American Economic Review* 62, 5, pp. 777-795.
- Boehm B.W., 1988, “*A Spiral Model Of Software Development And Enhancement*”, *IEEE Computer*, Vol. 21, N°5, May.
- Boehm B.W., Papaccio P.N., 1988, “*Understanding and Controlling Software Costs*”, *IEEE T.S.E.*, Vol. 14, N°10, pp. 1462-1477.
- Brynjolfsson E., 1993, “*The Productivity Paradox of Information Technology*”, *CACM*, Vol. 36, N°12, pp. 66-77.
- Brynjolfsson E., Hitt L.M., 1998, “*Beyond The Productivity Paradox*”, *CACM*, Vol. 41, N°8, August, pp. 49-55.
- Coase R., 1937, “*The Nature Of The Firm*”, *Economica*, Vol. 4, pp. 386-405.
- Dakhli S., 1998, “*Prototyping*”, Ph.D. thesis, Paris IX-Dauphine University, Paris.
- Dakhli S., Toffolon C., 1997, “*A Three Layers Software Development Method: Foundations and Definitions*”, *Third IEEE Conference on Engineering of Complex Computer Systems*, Como, Italy, Sept. 8-12, pp. 162-172.
- Gurbaxani V., Whang S., 1991, “*The Impact of Information Systems on Organizations and Markets*”, *CACM*, Vol. 34, N°1, pp. 59-73.
- Leavitt H.J., 1963, “*The Social Science of Organizations, Four Perspectives*”, Prentice-Hall, Englewood Cliffs, New Jersey.
- Malone T., Yates J., Benjamin R.I., 1987, “*Electronic Market and Electronic Hierarchies*”, *CACM*, Vol. 30, pp. 484-497.

- Nauer P., Randall B. (eds.), 1968, "*Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*", NATO Scientific Committee, Garmisch Germany, October, NATO Scientific Affairs Division, Brussels, Belgium, pp. 138-150.
- Picot A., Kirchner C., 1987, "*Transaction Cost Analysis of Structural Changes in the Distribution System: Reflections on Institutional Developments in the Federal Republic Germany*", Journal of Institutional and Theoretical Economics, Vol. 143, pp. 62-81.
- Pressman L., 1997, "Software Engineering: A practitioners Approach", McGraw-Hill.
- Simon H.A., 1983, "*Models of Bounded Rationality*", (2 volumes), MIT Press, Cambridge.
- Solow R., 1987, "We'd Better Watch Out", New York Times Book Review, July 12.
- Standish Group, 1995, "The Chaos Report".
- Toffolon C., 1996, "*Prototyping Incidence in Software Engineering Development Methodologies*", Ph.D. thesis, Paris IX-Dauphine University, Paris.
- Toffolon C., 1999, "*The Software Dimensions Theory*", in the Proceedings of ICEIS'99 Conference, Setubal, Portugal, to be published by KLUWER ACADEMIC PUBLISHERS in "Enterprise Information Systems", Selected Papers Book, Joaquim Filipe (Ed.).
- Toffolon C., Dakhli S., 2002, "A Global Model of Electronic Commerce", to be published in the Proceedings of ICEIS'02 Conference, Madrid, Spain, April 3-6, 2002.
- Wiederhold G., 1995, "*Mediation and Software Maintenance*", Technical Report, Stanford University.
- Willcocks L., 1994, "*Information Management: The Evaluation of Information Systems Investments*", Chapman & Hall, London, U.K.
- Willcocks L., 1999, "*Beyond the IT Productivity Paradox*", John Wiley & Sons, New York, U.S.A, 1999.
- Williamson O.E., 1981, "*The Modern Corporation: Origins, Evolution, Attributes*", Journal of Economic Literature, Vol.19, December, pp. 1537-1568.